3

Comprendre GNU/Linux, les distributions et les modèles de développement

Le Libre est affaire complexe : il évolue selon ses propres standards, sa propre organisation, ses propres modèles. De ce point de vue, Linux est un exemple parlant et de prime importance. Outre la compréhension nécessaire de ces particularités, appréhender ce qui fait la réussite, la puissance du Libre et de Linux en particulier peut être très intéressant pour l'entreprise souhaitant mettre en place des procédés innovants de développement dynamique.

Considérations générales sur le noyau Linux

Linux est un noyau et non un système d'exploitation. Afin de lever totalement l'ambiguïté, le terme de GNU/Linux est employé par les libristes quelque peu puristes, afin de signifier l'importance radicale, du point de vue du système complet, du projet GNU : au-dessus du noyau Linux tournent donc des applications, les plus fondamentales étant issues de GNU, qui en réalité font appel à des bibliothèques partagées pour interagir avec lui (notamment la bibliothèque C, *glibc*). Cette notation est en outre pratique dans l'absolu pour savoir si l'on parle du noyau seul ou d'une distribution.

Linux, en tant que tel, est donc un projet indépendant, dirigé par Linus Torvalds, déclaré "gentil dictateur". Il décide des évolutions futures dans les grandes lignes, mais aussi s'il faut accepter ou non de nouvelles fonctionnalités développées.

Contrairement au projet GCC, les droits d'auteur sont toujours détenus par les développeurs originaux, et non reversés. Linux "appartient" donc à tous ceux qui y ont contribué. En cas de litige, la partie du code concernée peut être soit simplement supprimée, soit entièrement réécrite. Évidemment, un tel projet possède une traçabilité sans faille. De même, toutes les anciennes versions du noyau sont disponibles sur le site de téléchargement et de versionnement : **http://kernel.org/**.

Le système de développement

Le système de codage est effectivement fondé sur le logiciel de versionnement de sources décentralisé, Git (développé par Linus Torvalds lui-même). Après avoir codé sur sa branche locale, le développeur soumet alors un *patch*, qui sera revu par les mainteneurs du noyau : idée de fonctionnalités (on n'ajoute pas de choses inutiles ou en double), recherche d'erreurs (*bugs*), et même style de codage (strictement défini) sont vérifiés. Tout le cycle de développement est public.

Un patch important (module entier, nouveau pilote, lourde modification de fonctionnalité) lorsqu'il est accepté est alors intégré à une branche "temporaire" du kernel : la *staging tree*. D'autres branches existent comme celle expérimentale, dite "-mm" et gérée par Andrew Morton. Après un certain temps passé dans cette phase de tests, des instructions peuvent être données pour que la contribution soit définitivement intégrée à la branche principale stable, dite "vanilla". Si le contributeur (développeur ou entité ayant effectué la soumission) ne répond pas aux attentes, son code est alors rejeté. Cela a pu arriver pour des milliers de lignes de code de Microsoft comme pour des fonctionnalités essentielles pour Android (figeant durablement le noyau du projet à la version 2.6.32, du début de l'année 2010). Sur son blog, Greg Kroah-Hartman retrace l'histoire de ces lignes de code retirées du noyau : **http://www.kroah.com/log/linux/android-kernel-problems.html**.

La rigueur prime toujours, et les mainteneurs principaux (Linus Torvalds ou Greg Kroah-Hartman, entre autres) se doivent d'être intransigeants pour assurer la viabilité du projet sur le long terme. Si une contribution est acceptée, elle est alors fusionnée (*merging*) à la branche principale : ce sera alors la communauté qui s'occupera de la maintenir, comme tout le reste du noyau. Cela signifie qu'en cas de modification interne ayant des effets de bords (changement de l'API, d'une structure, etc.), les répercussions engendrées seront automatiquement répercutées. Ceci est un avantage décisif poussant à faire accepter ses contributions ; dans le cas contraire, le cycle de développement devient en effet rapidement très lourd.

D'un point de vue macroscopique, Linux ne bénéficie pas d'un plan de développement précis. Pour paraphraser Linus Torvalds : *Linux is evolution, not intelligent design*. Des statistiques précises existent cependant. En 2010, le noyau Linux comporte 11,5 millions de lignes de code dans 28 000 fichiers source, le rythme de croissance allant en s'accélérant : les deux dernières années, c'est 2,8 millions de lignes de code qui ont été ajoutées. Ces cinq dernières années, plus de 5 000 développeurs pour plus de 500 entreprises ont participé au projet (aux premiers rangs desquelles on peut compter Red Hat, IBM, Novell ou Intel).

Référence

Pour consulter des statistiques sur le noyau : *Linux Kernel Development* de Greg Kroah-Hartman, SuSE Labs / Novell Inc., Jonathan Corbet, LWN.net, Amanda McPherson, The Linux Foundation, disponible sur **http://www.linuxfoundation.org/publications/whowriteslinux.pdf**.

Il sort environ une nouvelle version majeure de Linux tous les deux à trois mois. Le nombre de patches proposés entre deux versions tend à s'accroître, dépassant à présent les 10 000, alors que leur nombre tournait autour de 5 000 avant la version 2.6.24. Nous en sommes à présent à la version 3.0.4 (octobre 2011). Évidemment, le temps d'impression du présent ouvrage aura déjà rendu cette indication caduque.

Depuis quelques années, le développement du noyau suit une organisation réglée selon des intervalles de sortie plus ou moins réguliers (deux à trois mois). Chaque cycle débute par une courte fenêtre de fusion (*merge window*), d'environ deux semaines, durant laquelle les patches soumis par les développeurs sont appliqués (après avoir été sélectionnés et avoir passé tous les tests dans les branches parallèles instables). La seconde phase est celle de stabilisation et de débogage intensif, jusqu'à ce que la mouture soit jugée suffisamment au point pour sortir. Le *bootloader* U-Boot utilise une organisation similaire (voir **http://www.denx.de/wiki/u-boot/ReleaseCycle**).

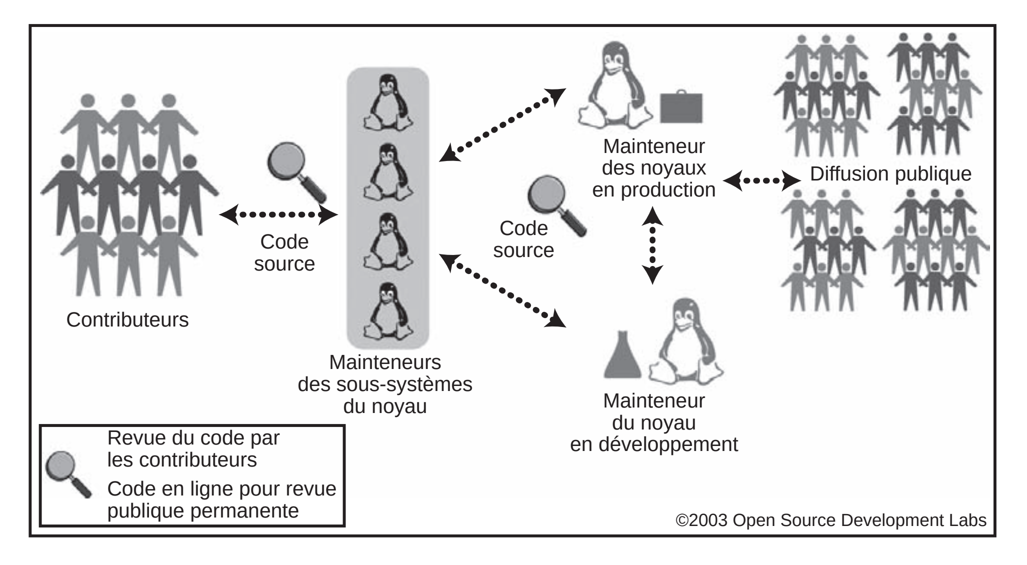


Figure 3.1

Organisation du développement du noyau Linux.

D'un point de vue de la numérotation des *releases* (moutures), il faut surtout noter le système de branche paire ou impaire : jusqu'au 2.6.39, la version du noyau est notée sous la forme de *x.y.z[.zz]*, dans lequel :

- *x* est la version majeure, "2" depuis 1996.
- *y* est la version mineure, en l'occurrence "6" depuis 2003 ; si ce numéro est impair, il s'agit d'une branche instable, tandis que s'il est pair, la branche est stable.

- *z* est le numéro de révision, incrémenté d'1 à chaque sortie ; on compte ainsi trente-neuf versions dans la branche 2.6.x.
- *zz* était employé dans le cas d'une sortie imprévue concernant une petite modification ayant tout de même nécessité une mise à jour anticipée (par exemple en cas de problème lourd sur un *driver* ou de *bug* gênant) ; à présent, ces sous-versions sont inscrites dans un plan de mises à jour rétro-compatibles de corrections mêmes mineures.

Cette numérotation a l'avantage d'être claire (elle est aussi complétée par un nom original, que l'on trouve en tête du Makefile principal), et peut inspirer celles d'autres projets particuliers. Elle n'est absolument pas répandue dans le monde du Libre, où les idées les plus folkloriques peuvent apparaître (les numéros de versions projet LaTeX tendent vers Pi, par exemple). Cependant, elle ne reflète pas les changements radicaux qui peuvent s'opérer d'une version à l'autre. Les versions 2.6.13, 2.6.16 (marquant le support de l'EABI sur ARM, et donc potentiellement d'une incompatibilité entre les versions du noyau précédentes et l'espace utilisateur, la rétro-compatibilité étant restée expérimentale depuis) ou 2.6.28 sont, par exemple, réputées pour avoir introduit des changements majeurs, qui ont entre autre rendu très difficile la migration de modules externes par les industriels (leurs pilotes spécifiques, en somme, qu'ils n'ont pas intégrés au noyau).

Depuis la mise en place du cycle de développement par fenêtre de fusion, les itérations de versions sont plus basées sur le temps que sur les fonctionnalités. Le troisième chiffre devenant important, Linus Torvalds a finalement décidé, le 29 mai 2011 (**http://lwn.net/Articles/445223/**), que la version succédant au 2.6.39 serait la 3.0 (un troisième chiffre optionnel pouvant marquer les versions stabilisées successives d'une même version) ; le noyau 3.0 est sorti le 21 juillet suivant. Ce changement de branche majeure n'a donc rien à voir avec une modification de l'architecture : c'est une simple renumérotation pratique. Elle intervient aussi pour les 20 ans du noyau.

Un nouveau système de "maintenance allongée" a fait son apparition pour répondre à ce besoin de stabilité de la part des industriels : l'incontournable Greg Kroah Hartman a ainsi annoncé en janvier 2010 (**http://www.kroah.com/log/linux/stable-status-01-2010.html**) qu'à l'image de ce qui se faisait déjà pour le 2.6.27, le 2.6.32 serait maintenu deux à trois ans (c'est-à-dire remis à jour par des *backports* de correctifs, autant pour les *bugs* que pour la sécurité), tandis que le pourtant récent 2.6.31 était pour sa part déjà abandonné. Ces versions sont dénommées *"long term" stable releases (LTS).* En moins d'un an, une bonne vingtaine de versions de la 2.6.32 sont ainsi apparues en page d'accueil du site *kernel.org,* avec un *log* complet des changements apportés.

On l'aura compris, de longues et nombreuses discussions sont nécessaires pour gérer un tel projet. Celles-ci s'effectuent sur la LKML (*Linux Kernel Mailing List*), dont l'abonnement est ouvert à tous. Les discussions sont publiques, et des synopsis des plus importantes ou intéressantes sont même régulièrement publiés sur le site

http://kerneltrap.org/, pour suivre l'actualité. Si tout un chacun peut y participer, il est absolument indispensable de respecter la netiquette (par exemple en répondant en dessous du mail cité). Froisser inutilement sur la forme ou le fond, ou générer du bruit inutile, équivaut à perdre toute crédibilité auprès de la communauté.

Il faut aussi faire attention, comme dans toute société, aux personnalités importantes dont l'avis compte davantage (citons Greg Kroah Hartmann ou Alan Cox). Le ton sur la liste est vif, et certaines tirades "inspirées" de Linus Torvalds sont devenues cultes : le monde du Libre est celui du narcissisme, et mieux vaut être bien équipé pour l'affronter. En tant que nouveau venu, faites vos preuves et montrez toujours patte blanche.

Avantages et inconvénients de l'utilisation de Linux pour l'embarqué

Une question se pose : pourquoi Linux en particulier et pas autre chose ? Les avantages sont tout d'abord ceux du Libre, que nous avons déjà évoqués :

- La licence, GPLv2, et l'organisation du projet libre sont garants de pérénité et d'une forte évolutivité.
- Il n'y a pas de *royalties* à payer.
- Les sources sont gratuites, pour toute utilisation ou modification.

Il y a ensuite les points forts techniques, qui en l'occurrence sont très nombreux :

- L'empreinte mémoire est faible (de l'ordre de 4 Mo) en considération des capacités techniques.
- Le code est efficace, optimisé, orienté pour de fortes performances (fonctionnant sur serveurs ou sur supercalculateur), et très stable ; l'*uptime*, c'est-à-dire le temps entre deux redémarrages du système, est par exemple l'un des meilleurs au monde.
- Le noyau possède par son architecture un espace utilisateur (*userspace*) efficace, il garantit un faible *overhead*, assez indépendant pour donner la possibilité de changer de noyau sans avoir à modifier les applications, et propose notamment le meilleur support par un système d'exploitation de Java sur une architecture embarquée.
- L'architecture est modulaire : reparamétrable à chaud, seule la mémoire utile est occupée par le noyau ; paramétrable très finement, les fonctionnalités sont alors activables ou désactivables, avec une gestion des dépendances.
- Le noyau présente une adaptabilité temps-réel mou par défaut, que l'on peut rendre compatible avec des contraintes de temps-réel dur sans trop de difficulté.

- Le noyau dispose d'un énorme support matériel (en nombre de *drivers* comme en termes d'architectures matérielles) et une grande portabilité (c'est-à-dire le fait de pouvoir faire fonctionner le code sur de nouvelles architectures).
- Il offre la possibilité de gérer les architectures multiprocesseurs SMP (*Symetric MultiProcessing*) ; pour une application dans le cadre du temps-réel, voir l'article : **http://www.linuxjournal.com/article/9361**.
- Le noyau dispose d'un ordonnanceur, par défaut en O(1) ou depuis la version 2.6.23 (2007) en O(log n), assurant de base une excellente maîtrise de la charge logicielle ; l'ordonnanceur (*scheduleur*) et le noyau sont préemptibles (sous quelques réserves) pour la branche 2.6.x, assurant une grande réactivité.

Note

Écrit par Ingo Molnar, aussi à l'origine de la majeure partie des fonctionnalités temps-réel du noyau, le *Completely Fair Scheduling* (CFS) est fondé sur un arbre rouge-noir, et non une file de processus.

- Beaucoup d'outils et de bibliothèques compatibles avec le noyau POSIX sont disponibles (quasiment toujours gratuitement).
- La plateforme de développement est identique à la plateforme cible : on développe sur du Linux pour du code à destination de Linux, seule l'architecture matérielle change potentiellement (ou peut changer en cas de compilation croisée). Il est alors tout à fait possible de développer un logiciel pour x86 et d'assurer le portage sur ARM ponctuellement, par exemple, chose absolument inédite jusque-là dans l'embarqué. Précisons que l'architecture de Windows CE n'a guère de rapport avec les moutures homonymes que l'on trouve sur le bureau.
- La plateforme est très favorable au développement, citons : le compilateur GCC, le système de Makefile (fichier d'instructions de compilation), les *autotools* (génération de Makefile complexe), le format de compression du code source *tar.gz* ("tarball"), les gestionnaires de versionnement de sources (CVS, SVN ou encore Git), et enfin le support de tous les langages de programmation.
- L'architecture est héritée des systèmes *NIX : "tout est fichier", les droits utilisateur (système de droits pour l'utilisateur – uid –, le groupe – gid – et le reste du monde ; l'administrateur se nomme *root* et a tous les droits), le système de pipe et l'organisation en centaines de petits utilitaires basiques, même gestion des variables d'environnement propres à chaque processus et héritées lors du *fork*, ou encore le système de hiérarchie des fichiers et le montage.

Note

Comme il ne s'agit pas toujours à proprement parler de descendants direct d'UNIX, comme c'est le cas pour Linux, on note les *UNIX-like*, "*NIX".

- On trouve nativement les supports pour TCP/IP (pile éprouvée sur les plus grands serveurs de la planète), le Bluetooth, le multimédia (V4L et VAL2 – *video for Linux*), le multi-cœur, etc.

Ainsi, de par ces caractéristiques, le système peut facilement être mis à jour.

Linux a été la solution idéale qui s'est présentée au bon moment, dès les années 2000, et surtout depuis 2004. Le besoin de nouvelles fonctionnalités plus riches et modulaires apparaissant chez les industriels intégrateurs en même temps que la volonté de se détacher des anciens modèles très contraignants de licences se faisait sentir, l'adoption de cette technologie pourtant mal maîtrisée par un milieu peu réactif et traditionnellement assez conservateur, n'a pris que quelques années, à l'image d'un VxWorks dans les années 1990.

Au sein des équipes, une certaine information s'est propagée sur les bénéfices à tirer de Linux ; cependant, les revers de la médaille sont souvent mal connus ou trop sous-estimés. De là des glissements de planning fâcheux, voire de lourdes déceptions lorsque les ressources mobilisées n'ont pas été suffisantes (ou se sont révélées superflues) pour obtenir quoi que ce soit de fonctionnel. Nous avons déjà noté les inconvénients intrinsèques au Libre pour l'industriel. Plus spécifiquement avec Linux, il faut porter son attention sur les points suivants :

- Le code est complexe, comportant de très nombreuses couches d'abstraction (nécessaires à la portabilité) où il est nécessaire de savoir naviguer pour apporter des modifications.
- La compilation est assez longue et le développement pour un nouvel entrant impose un apprentissage complexe (des formations existent mais ne peuvent couvrir tous les cas, tandis que la littérature est assez réduite).
- Les dépendances entre logiciels et bibliothèques peuvent rendre le système très instable si sa construction a été mal gérée.
- Les outils de débogage, longtemps honnis par Linus Torvalds, sont encore basiques en comparaison des solutions propriétaires pour les industriels.
- La configuration pour les besoins spécifiques du noyau n'est pas une tâche triviale et s'apparente quelque peu à la taille d'un bonsaï.
- La réadaptation des sources ou de l'architecture pour une compatibilité temps-réel nécessite une grande maîtrise de la solution.

De plus en plus de revendeurs de matériel (si ce n'est la totalité) proposent un support Linux (dans le *Board Support Package*), évitant à l'intégrateur final les souffrances d'une phase de réadaptation du noyau. Cependant, ils ne fournissent souvent que quelques versions spécifiques, parfois très âgées et ne répondant pas à la problématique particulière du projet, ce qui impose alors de faire par soi-même le portage. Les mauvaises surprises peuvent alors rapidement se multiplier, puisque

le passage de modules entre deux versions éloignées peut être tout simplement impossible.

Note

Un module ou *driver* est fortement lié à une version donnée du noyau Linux : s'il est fourni sous forme de sources isolées, il faudra retrouver les sources complètes du noyau associé pour le compiler ; s'il est fourni sous forme binaire, impossible de l'utiliser avec une autre version du noyau que celle avec lequel il a été compilé.

Si la mésaventure est très formatrice – voir ma conférence sur les problèmes rencontrés lors du portage d'un même pilote réseau sur deux versions différentes du noyau : **http://www.gillesblanc.com/content/conférence-rts-2010** –, elle est cependant potentiellement catastrophique en termes de gestion de projet, pouvant même impliquer dans de rares cas une solution de contournement par paravirtualisation, plutôt qu'un très coûteux portage. Il est donc absolument nécessaire de se renseigner sur le support offert par le constructeur et le revendeur matériel avant l'achat de la plateforme cible.

Les distributions Linux

Il est nécessaire en tout premier lieu d'avoir une bonne vue d'ensemble d'un système GNU/Linux (voir Figure 3.2).

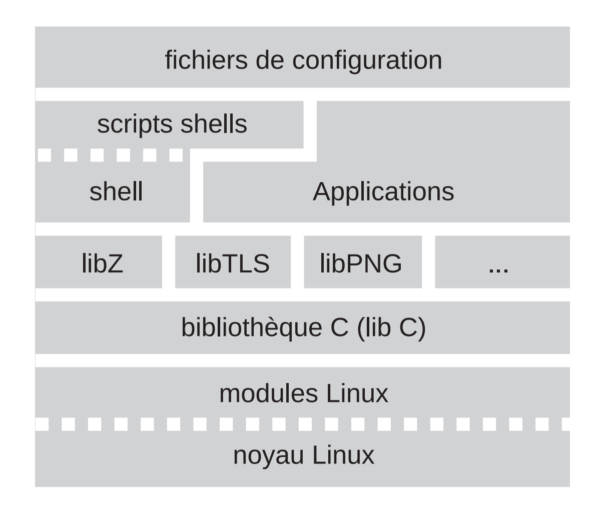


Figure 3.2
Organisation d'un système Linux.

Côté utilisateur, les applications assurent les interactions fonctionnelles. Il s'agit de programmes, compilés à partir de sources. La puissance du modèle UNIX repose sur l'utilisation de bibliothèques partagées (fichiers en extension *.so*). Une bibliothèque est une collection de fonctionnalités appelables par un programme. Concrètement, à la compilation, l'API (*Application Programming Interface*) de la

bibliothèque est renseignée par l'inclusion d'un fichier d'*include* (extension *.h* en langage C) et par le nom de la bibliothèque à lier dans la ligne de commande du compilateur. Aussi, les bibliothèques peuvent potentiellement s'appeler entre elles : une homogénéité doit donc être conservée pour que le système fonctionne. Ce système est à différencier des bibliothèques dynamiques sous Windows (les fameuses "dll") dont l'appel est très souvent manuel, et qui sont pour la grande majorité intrinsèquement liées à un programme particulier.

Sous Linux, la résolution d'appels est automatique (c'est le format ouvert de programme exécutable ELF qui renseigne les dépendances). De même, la bibliothèque C (*libC*, la version GNU étant la *glibc*), assurant l'ensemble des appels systèmes au noyau (sous peine de devoir le faire en assembleur, ce qui n'est guère envisageable), sur laquelle repose tout programme d'un système GNU/Linux, est commune à tous les exécutables d'un système Linux donné. Cette unicité a pour conséquence un gain de place et une simplicité d'architecture importants. En revanche, l'homogénéité du système implique qu'un programme non compilé contre la bibliothèque C disponible aura de très grandes chances de ne pas fonctionner.

Une distribution GNU/Linux est une version particulière du système d'exploitation. La manière d'assembler les programmes et bibliothèques au-dessus du noyau, ainsi que leurs versions respectives, les systèmes de configuration, le système d'installation et le système de paquets employé déterminent une distribution. Une particularité du monde Linux vient du fait que les développeurs du logiciel sont quasiment toujours indépendants des intégrateurs logiciels : les premiers codent, les seconds créent des paquets pour les distributions.

La problématique principale se rapporte au paquetage. Un paquet est un ensemble, sous forme de fichier-archive, de binaires, de scripts, de fichiers de configuration, munis d'instructions d'installation, pouvant être interprété par un gestionnaire de paquets afin de procéder à l'installation d'un logiciel, d'un module (noyau ou applicatif), ou de fonctionnalités diverses, voire du noyau lui-même. On distingue aussi les paquets source, qui permettent d'obtenir les sources du logiciel et de le compiler automatiquement pour obtenir le même résultat d'installation (outre pour le respect de la GPL, certaines distributions dites "source" ne disposent que de ce type de paquets), et les paquets de débogage (notés avec l'extension *-dbg*) associés aux paquets "standard", qui permettent d'obtenir les instructions nécessaires pour effectuer le déverminage (comme on dit en bon néo-français) du logiciel concerné.

Les gestionnaires de paquets gèrent les dépendances et les conflits. Les paquets doivent être vus comme des entités de fonctionnalités. Il se crée donc un réseau de dépendances entre un paquet logiciel et les autres bibliothèques ou logiciels qu'il utilise, eux-mêmes fournis sous forme de paquets, qui peuvent à leur tour avoir leurs propres dépendances. Installer un certain paquet impliquera donc d'installer ses dépendances, sous peine d'empêcher son fonctionnement (se pose alors la problématique de la désinstallation des dépendances orphelines ; différentes politiques sont possibles et se valent).

De plus, certains logiciels dépendent d'une bibliothèque d'une version donnée, tandis que d'autres utilisent une autre version de la même bibliothèque : s'il est possible de faire cohabiter les deux, tout va bien ; sinon, il y a conflit de paquets, il est impossible d'installer l'un sans désinstaller l'autre. La bonne gestion des dépendances et des conflits est à la base de la réputation d'une distribution. C'est aussi un lourd problème qu'il faut savoir à son tour gérer lors de la construction d'une distribution pour l'embarqué.

Les distributions sont très nombreuses ; le site distrowatch (**http://distrowatch.com/**) en compte une centaine principales. Si beaucoup sont anecdotiques, certaines sont très importantes.

- **Slackware** : première distribution (1993) mais aujourd'hui très peu présente, elle a cependant posé les bases qui sont toujours en cours dans la gestion des projets de distributions.
- **Debian** : entièrement communautaire, elle a donné lieu à de nombreux *forks* (Ubuntu, Mepis, etc.), on la trouve essentiellement sur des serveurs ; son gestionnaire de paquets est *apt* (format *.deb*).
- **Ubuntu** : héritant de Debian ses grands principes, son aspect plus à la mode (techniquement plus mise à jour, et bénéficiant d'un fort marketing de la part de la société Canonical Ltd du milliardaire sud-africain Mark Shuttleworth, à l'origine du projet) l'a propulsé très rapidement comme la distribution la plus installée ; on la rencontre de fait fréquemment sur les postes de développement.
- **Red Hat** (RHEL) : la société du même nom a été la première à éditer une distribution Linux supportée sous contrat, offrant une assurance aux industriels, qui l'ont massivement installée ; on la trouve donc sur les postes de travail des grands groupes industriels ; Red Hat utilise des paquets RPM et le gestionnaire de paquets homonyme, auquel a succédé *yum*.
- **Fedora** : c'est le pendant communautaire de la Red Hat. Certaines distributions ont été dirigées uniquement en interne par des sociétés : immanquablement, elles n'ont guère tenu plus de deux ans. Fedora est utilisée par le grand public, ou par les industriels lorsqu'ils ne souhaitent pas payer de support Red Hat. Ils gagnent ainsi un accès complet aux paquets d'une distribution compatible. Notons aussi que la distribution totalement libre CentOS propose les mêmes paquets recompilés que la Red Hat, assurant une parfaite compatibilité binaire pour chaque version de la distribution ; on peut donc récupérer des paquets CentOS pour procéder à une installation sur Red Hat (par exemple du compilateur, non inclus par défaut).
- **Mandriva** : issue du projet Red Hat, cette distribution historique du paysage français a été gérée par la société homonyme. Elle utilise aussi des RPM (non-compatibles avec ceux de Red Hat). Il n'est pas impossible de rencontrer cette distribution dans l'industrie (où l'entreprise s'est quelque peu déployée, notamment *via* un projet sur architecture MIPS), même si le projet oscille

régulièrement entre la mort clinique et la résurrection miraculeuse au fil des rachats. De fait, un *fork* communautaire d'anciens employés a été créé : **Mageia**.

- **SLED**, **SLES**, **OpenSUSE** : racheté par Novell, la distribution se veut à la fois très accessible par le grand public (disposant de la version communautaire OpenSUSE) et par les entreprises (versions *Enterprise Desktop* ou *Enterprise Server*), de telle sorte que certains grands comptes de l'industrie l'utilisent sur leur poste de travail ; on distingue le puissant gestionnaire de configuration YaST de l'installeur de paquets *RPM* `zypper` ; un portage est disponible sur plusieurs architectures, *via* des outils de compilation. OpenSUSE est en outre la distribution que je vous recommande à titre personnel.
- **Gentoo** : fondée sur une idée héritée des systèmes BSD, il s'agit d'une distribution source, c'est-à-dire où tout paquet logiciel est téléchargé sous forme de sources, et compilé sur la machine (avec toutes les options d'optimisation possibles) ; elle est très appréciée dans certains milieux professionnels de développement embarqué.
- **LFS** : *Linux From Scratch* est à la limite de ce que l'on appelle une distribution puisqu'il s'agit de compiler et intégrer soi-même, en suivant le mode d'emploi fourni, la distribution de bout en bout, sans s'aider de paquets automatisés ; elle donne une idée du travail monstrueux nécessaire pour construire proprement un système d'exploitation GNU/Linux complet ; notons que le projet **CLFS** (*Cross Linux From Scratch*) est un LFS pour architecture différente de celle de l'hôte, avec pour objectif d'être embarqué (voir **http://trac.cross-lfs.org/**).
- Les distributions **lives** (Knoppix sur CD/DVD, Damn Small Linux sur clef USB) : Linux peut démarrer sur tout type de support, y compris des supports amovibles ; professionnellement, ces distributions sont très pratiques pour tester du matériel, effectuer des récupérations ou simplement avoir une idée de la manière de mettre en place un système calibré et en lecture seule (avec extensions envisageables en écriture).

Ces distributions partagent des points communs plus ou moins forts en fonction de leurs généalogies. Un effort de standardisation a été mis en place *via* la compatibilité LSB (*Linux Standard Base*), pour que les paquets puissent être utilisables dans une certaine mesure d'une distribution à une autre, et que les utilisateurs puissent s'y retrouver dans l'arborescence du système (normalisée selon la FHS – *File System Hierarchy Standard*). Mais la force de Linux repose en réalité sur leurs différences (de philosophie, d'approche technique, de cycle de *release*, de politique de sécurité, de fréquence de mises à jour, etc.), qui donnent libre choix à l'utilisateur (en l'occurrence le développeur de Linux embarqué) de trouver ce qui lui convient le mieux. Les influences dans l'embarqué et le temps-réel ne sont pas négligeables.

La philosophie d'extrême portabilité de la distribution Debian (dont la version ARM est compilée nativement !) lui a ainsi permis d'être embarquée telle quelle

sur de nombreux projets en production. De même, l'historique de Red Hat et Fedora explique que l'on rencontre cette dernière distribution communautaire sur des projets militaires en production, sur plateforme x86.

Un *business model* émergeant : les collaborations entre commercial et communautaire

Il existe trois principaux *business models* pour une société voulant développer son chiffre d'affaire autour du logiciel libre :

- L'expertise et le service : faire valoir sur un marché spécifique des compétences supérieures à la concurrence.
- L'édition : éditer son propre logiciel libre – qu'une société tierce peut tout à fait installer (au minimum pour tester), revendre ou *forker* – , avec l'idée de monter une offre commerciale autour de services associés, que ce soit de l'installation ou des ajouts de fonctionnalités spécifiques, qui seront à la charge d'un client particulier et pourront dans le second cas bénéficier à tous.
- Le "freemium" ou la double licence : on tombe sur un schéma semi-propriétaire, où une partie du logiciel peut être libre mais les fonctionnalités avancées payantes, ou alors où une utilisation fermée du code se monnaye auprès de l'éditeur (ce qui implique qu'il garde pour lui tout le *copyright* : le développement ne peut être communautaire).

Dans le monde de l'embarqué, cela s'est traduit de la même manière :

- Les sociétés de services proposent une expertise à la prestation ou au forfait autour de Linux embarqué (schéma classique).
- Les sociétés éditrices (Wind River, Sysgo, Denx, OKLabs, etc.) proposent une offre de Linux embarqué libre, sous forme d'une distribution particulière (ou un *bootloader* libre ou un hyperviseur de paravirtualisation libre, etc.), mais vendent un service d'adaptation à une problématique particulière, en faisant valoir leur expertise autour de leurs produits.
- Ces mêmes sociétés éditrices peuvent vendre des outils particuliers rendant de grands services, mais ils sont fermés et soumis à licences propriétaires. Elles peuvent encore proposer une version plus améliorée que celle libérée (datant souvent d'un an).

Note

On dit d'un logiciel qu'il est libéré s'il est redistribué, par les détenteurs de son droit d'auteur, sous une licence libre, tandis qu'il était soumis auparavant à une licence de type propriétaire.

Cependant, ces modèles montrent vite leurs limites dans le cadre de projets d'envergure. En effet, si la vente de services ne peut s'appliquer qu'à des projets particuliers, dont le reversement à la communauté est souvent le dernier des soucis (c'est dans ces cas-là, majoritaires, que le nombre de manquements au respect des licences libres est le plus important), le modèle "semi-libre" est inadapté à une large diffusion du code, et donc à sa reprise par une communauté. Par exemple, la double licence libre/propriétaire implique que tout développeur voulant améliorer le logiciel concerné cède ses droits d'auteur à la société qui revend ensuite son travail. Ou encore, l'amélioration d'un logiciel datant de plus d'un an, avec souvent une documentation non rendue publique (puisque vendue), n'est guère encourageant.

Or, ce qui fait la puissance tant convoitée du Libre est tout simplement la communauté, soit d'un point de vue économique une ressource forte de développeurs normalement chers, et pour le coup tout à fait gratuits. Une communauté se forme habituellement de manière spontanée, regroupant des hackers plus ou moins expérimentés autour d'un intérêt commun. Il est très intéressant, pour une société, de pouvoir disposer d'une telle puissance intellectuelle : le développement de fonctionnalités supplémentaires gratuites peut s'avérer être un argument de vente auprès des connaisseurs (qui alimentent alors la communauté), et il ne reste qu'à récupérer les meilleures idées pour les réintégrer dans les nouvelles versions du produit (sous forme de mises à jour ou en usine).

Note

Du côté des NAS, par exemple, on compte une communauté auto-formée autour du Western Digital MyBook World (**http://mybookworld.wikidot.com/**) et une autre pour le Netgear Stora (**http://www.openstora.com/**). Pour les *set-top boxes*, il existe la FreedomBox Foundation (**http://www.freedomboxfoundation.org/**). Le projet OpenWRT s'est formé autour du routeur Linksys WRT54g ; cette distribution Linux embarquée est à présent utilisée dans bon nombre de produits commerciaux comme la neufbox ou la Fonera.

Pour encourager la naissance d'une communauté autour d'un projet commercial, il suffit d'être transparent (respect des licences, documentation, fourniture d'informations, etc.), d'avoir un système de mise à jour simple, d'éviter tout ce qui peut rebuter les *hackers* (comme les systèmes de signature par DRM). Mieux encore, pour lancer une communauté afin de profiter d'un écosystème riche autour de son produit, il faut de plus mettre en place une structure particulière (site web dédié, wiki, *howto*, *mailing list*, etc.), fournir l'intégralité des sources, des outils de compilation et de flashage (avec les méthodes de récupération du matériel en cas de problème), mais aussi mener l'organisation (depuis les développements jusqu'à l'animation de rencontres, en passant par le *brainstorming*), formaliser les cycles de développement sous peine d'en perdre la main et surtout ne jamais décevoir ceux qui participent au projet et donnent de leur temps. On peut aussi simplement leur offrir des T-shirts ! OKLabs ou TI l'ont bien compris : la reconnaissance est la base du management.

Les industriels ont un temps tenté de reconstituer cette forme de communauté à travers des consortiums : on revoyait alors les mêmes groupes mondiaux (Samsung, Motorola, etc.) s'allier à travers LiSP (*Linux Phone Standards*), l'OHA (*Open Handset Alliance*), l'OMA (*Open Mobile Alliance*), LiMo (*Linux Mobile*), etc. Rapidement, des doublons sont apparus, qui ont parfois fusionné (alignement de LiSP et OMA en 2007, adhésion de LiSP à LiMo en 2008). Même si l'un des buts affichés par ces organismes est la définition de standards communs, les projets qui en sont sortis durant des années ont été aussi peu nombreux que visibles, essentiellement concentrés sur le marché asiatique.

Dans un même temps, d'autres industriels ont tenté une approche différente, consistant à débloquer des ressources humaines consacrées à la participation de projets libres pour l'embarqué. Cette méthode est utilisée assez largement par de grands groupes du logiciel, mais dans le monde industriel cela reste difficile : les problèmes administratifs, les problèmes techniques (avoir accès à une connexion Internet disposant de tous les ports ouverts peut être un problème sans fin) et de communication interne propres à ces grands groupes rendent le mouvement assez marginal. Dans l'ensemble, ce sont plutôt des sous-traitants mandatés qui opèrent ce genre de participation, qui dès lors se fait en pointillés.

Quelques projets libres portés par des sociétés privées ont cependant vu le jour, à partir de 2006 essentiellement. Il s'agit pour le plus important de Maemo par Nokia, l'environnement graphique abouti pour ses mini-tablettes Internet N770, N800 et N810 et sur smartphone N900. On peut aussi citer Fon et son routeur WiFi Fonera, dont le *firmware* est libre, et qui a monté une communauté de développeurs en se greffant sur le projet OpenWRT. Cependant, ces projets n'ont pas réellement suscité un enthousiasme débordant et sont restés cantonnés dans leur utilisation à leurs perspectives initiales. Pis encore, le projet OpenMoko, lancé par la petite société taïwanaise FIC, très ambitieux (téléphone au *hardware* et *software* entièrement libres – à l'exception de ce qui est couvert par NDA, soit le GPS et la pile GSM –, avec kits de développement commandables en ligne), n'aura donné au bout de cinq ans que deux modèles de téléphones (Neo1973 et FreeRunner), dont les fonctionnalités ne sont guère utilisables. Ce projet est resté cantonné à un nombre restreint de passionnés.

Cet échec (même si le projet vit encore), dû à un éparpillement dans différentes directions (il existe trois principaux environnements graphiques, incompatibles entre eux !) et à un manque de fort soutien industriel, montre les limites d'action de petits acteurs manquant d'organisation. De même, la GP2X (et son successeur en 2009, la GP2X Wiz), projet abouti de console de jeu portable libre (à l'inclusion près de DRM pour assurer la fermeture de jeux propriétaires, et ainsi rassurer leurs éditeurs), porté par la société GamePark Holdings, est aussi restée confidentielle.

Peut-être poussés par la concurrence (notamment l'apparition de l'iPhone d'Apple), ou signe de l'aboutissement d'une réflexion interne, les consortiums ont accédé à une nouvelle maturité depuis deux ou trois ans, avec des alliances intégrant des experts du logiciel libre montrant une certaine aspiration envers l'embarqué. Non

sans quelques péripéties. C'est ainsi que Moblin (**http://moblin.org/**) a été créé par Intel sur Ubuntu avant de migrer sur Fedora, puis que le projet a été cédé à la *Linux Foundation*, avant de fusionner en janvier 2010 avec le projet Maemo de Nokia, pour donner MeeGo (abandonné depuis fin 2011 pour Tizen). De son côté, l'abandonné Ubuntu, *via* sa société éditrice Canonical, a formé avec le concepteur de CPU ARM, *Linux on ARM* (**http://linux.onarm.com/**), qui reste somme toute assez confidentiel, limité à quelques développeurs dans le monde, et un nombre réduit de cibles (asiatiques). Maemo (et donc MeeGo) ou *Linux onARM* possèdent un point commun : s'adresser au marché des MID, *Mobile Internet Device*, dont les *tablets* sont les successeurs. C'est aussi le cas, de manière involontaire, de MeeGo par Nokia et Android par Google, qui ont présenté jusqu'en 2010 deux manières très différentes d'adresser un même marché de la téléphonie. L'étude des deux stratégies sous-jacentes est riche d'enseignement.

L'émergence d'un nouvel acteur incontournable : Android

Un système libre s'est distingué en termes de relations couronnées de succès entre industriels et communauté : Android. Le grand gagnant aura été le dernier arrivé, Google, avec une plateforme développée en interne et en secret, avant d'être dévoilée à des industriels montrant patte blanche, puis d'être totalement libérée, formant alors une communauté. Si la mixture a pris, c'est que tous les ingrédients ont été savamment introduits :

- Création d'une attente et d'un *buzz* (technique commerciale de communication dont Apple est habituellement le spécialiste), avec des annonces non-officielles et autres rumeurs de "Google Phone" (ou "gphone") dès fin 2007. En fait, les seuls téléphones de la marque seront de la gamme Nexus, à partir de 2010, fabriqués par HTC, puis par Samsung, et présentant un seul intérêt distinctif : plus d'ouverture pour le développement logiciel applicatif sur les premières moutures.

Info

Le Nexus, sous-traité à HTC, peut être considéré comme un échec commercial. À la mi-2011, Google a fait l'acquisition de la branche mobile de Motorola, pour 12,5 milliards de dollars, mais la volonté de garder l'entité comme stratégiquement séparée a fait penser à certains analystes qu'il s'agit là d'une stratégie de protection vis-à-vis des problèmes de brevets logiciels, plutôt que d'une réelle volonté de produire du matériel à l'image d'Apple. Les concurrents de Motorola et intégrateurs d'Android ont d'ailleurs accueilli avec enthousiasme ce rachat.

- Annonce officielle d'un produit fini et totalement fonctionnel, ne nécessitant qu'une phase de tests et d'intégration, sous une licence libre, mais non encore disponible.
- Contacts avec des industriels choisis (forme de sélection attirant ceux qui veulent être élus, et flattant ceux qui le sont déjà), et développement d'un

réseau (*via*... un consortium) comme base d'un déploiement programmé sur un large nombre d'appareils.

- Sortie des appareils en même temps que la libération du code s'effectue, en octobre 2008 (soit trois mois après la création de l'App Store pour iPhone, lui-même mis en place un an après la mise sur le marché de l'appareil), et surtout que des méthodes d'intégration et développement pour la plateforme apparaissent, avec une documentation complète (Apple en revanche demande 99 $/an).
- Prise en main par les utilisateurs (forcément plus nombreux que pour un simple appareil particulier) et parmi eux par des développeurs, qui agrègent leurs propres créations à forte valeur ajoutée (non sans doublons), créant une communauté dynamique quoique dispersée.

Google a pris un pari risqué, notamment en laissant la concurrence d'Apple s'étendre (sachant que la base de terminaux est tout de même réduite au seul iPhone). Cependant, avec 30 000 applications, dont une partie sous licences libres (confondue dans l'ensemble des logiciels gratuits mais fermés, vivant souvent de bandeaux publicitaires), en seulement quelques mois, le retard semble déjà comblé. Ils ont aussi pris une seconde posture très risquée : le produit étant de nature logicielle et fortement évolutif, il est inconcevable de distribuer au grand public des logiciels n'ayant pas subi un minimum de certification. En effet, jusque-là, les applicatifs sur téléphones portables tournaient dans une machine virtuelle Java très restreinte en termes de droits ; de fait, leurs capacités étaient aussi limitées que leur intérêt. En ouvrant de nouvelles perspectives, on affaiblit *de facto* la sécurité : les applications peuvent communiquer à l'extérieur, lire les données sensibles contenues sur le terminal, écrire sur sa mémoire additionnelle, etc.

Note

On trouve le chiffre de 150 000 applications en juin 2010, deux fois plus pour iPhone. La pertinence de ce qui est nommé "application" n'apparaît en revanche jamais dans cette absurde comparaison de chiffres.

Le modèle d'Android est donc recentré à ce niveau sur le "Market", où les applications ayant reçu l'aval de Google peuvent être recherchées, téléchargées et installées simplement sur l'appareil[1]. Ce système est clairement inspiré des *repositories* des distributions Linux, mais ajoute à la recherche un système de notation et de commentaires des applications (c'est-à-dire un système de revue communautaire, assimilable à du *crowdsourcing*), couplé à un mécanisme de paiement pour une partie des applications. La limite du Market est en réalité due à l'intégration d'Android sur les terminaux et à la nature du code, libre ou propriétaire : en effet, les applications pèsent la plupart du temps entre 1 et 3 Mo et génèrent des données,

1. Cependant, un cas de logiciel malveillant malencontreusement diffusé sur le *Market* s'est vu désinstallé des terminaux Android par une opération distante, nonobstant l'absence de toute demande d'accord des utilisateurs...

qui peuvent être enregistrées sur carte SD additionnelle, mais pas toujours (ce qui peut représenter un risque d'usure prématurée de la flash interne). Le problème vient alors du manque de place pour le stockage des programmes (notamment lors de leurs mises à jour). Depuis la version 2.2 d'Android, une option de configuration propose ainsi de déplacer les programmes sur la carte SD de l'appareil, *via* un système de boucle locale chiffrée, afin d'interdire la récupération de code sous licence propriétaire. Cependant, seule une minorité d'applications semble supporter cette option : on touche là à une limite de la flexibilité du système en termes de cohabitation au sein d'un même appareil entre code libre et propriétaire, qu'il soit inclus par défaut ou téléchargé de l'extérieur.

Afin d'avoir accès au Market, le terminal doit lui aussi être estampillé par Google : c'est une des facettes principales de leur business *model* original qui, en se concentrant sur l'aspect applicatif des terminaux mobiles, impose un environnement proposant par défaut leurs services, tels que la recherche *via* leur moteur ou encore Google Maps, avec comme but de monnayer de la visibilité publicitaire pour les entreprises (Apple leur a d'ailleurs emboîté le pas dans le rachat de régies spécialisées pour appareils mobiles). Cependant, cela n'est pas sans risque vis-à-vis de la construction d'une communauté, qui peut se sentir corsetée (voire exploitée). D'autant que le modèle change. Le développeur devient le *packager* : il doit aussi prendre en charge le reformatage de son application et les tests, pour sa diffusion, et doit passer par le Market, contrôlé. Il est possible d'installer un programme directement, mais la procédure est d'une complexité aussi anodine pour un développeur qu'elle rebute la quasi-totalité des utilisateurs.

De plus, si le Market est tout à fait compatible avec le logiciel libre, contrairement à l'AppStore (Apple Store) et au Marketplace de Microsoft, il n'en reste pas moins que le contrôle des applications est effectué en fonction de leur "morale".

Note

La version de VLC sur iPhone et iPad a été retirée sans préavis après qu'un des développeurs s'est fait écho des inquiétudes concernant le règlement très restrictif de la plateforme de téléchargement d'Apple : il s'agit d'une restriction dans les *User Agreements* quant à la diffusion du code, en contradiction forte avec la GPLv2.

La réaction de la FSF est lisible à cette adresse : **http://www.fsf.org/blogs/licensing/more-about-the-app-store-gpl-enforcement**.

Ainsi, il est totalement interdit sur les deux plateformes de distribuer des contenus pornographiques[2]. En revanche, la différence entre les deux conceptions des appareils est fondamentale : sur les produits d'Apple, le contrôle s'effectue *a priori*, et

2. On se souvient que le logiciel libre, tel qu'il est pensé, ne peut imposer de restriction d'usage, et qu'il est plus dommageable, *a priori*, d'embarquer du logiciel libre dans des missiles que de s'en servir pour admirer la plastique dénudée de ses semblables. Ce type de restriction, aussi légitime soit-il, dans la mesure où il va au-delà de ce qu'interdit la loi, ne cadre pas avec l'esprit du Libre et du principe de neutralité.

seule une application signée peut être installée ; avec Android, toute application peut être installée, indépendamment (pour tester un développement, par exemple) ou depuis un portail tiers. Pionnière habituelle, l'industrie pornographique a ainsi été la première à ouvrir son Market alternatif (MiKandi). Depuis quelques temps, on voit apparaître un véritable business model autour de la fourniture de Market à destination du déploiement et de la gestion de parcs d'appareils mobiles pour les sociétés : elles peuvent ainsi propager des applications particulières sans avoir à les publier ni à solliciter l'avis de Google (qui laisse la liberté de publication, mais se garde aussi la possibilité de retrait à tout moment du Market – une sorte de modération *a posteriori*).

Dans le *Windows Phone Marketplace Application Provider Agreement* (**http://create.msdn.com/downloads/?id=638**), on trouve le point 5.e excluant toute licence de type Open Source avec obligation de publication, listées en 1.l ("Excluded License"), notamment la GPLv3 citée explicitement (une bibliothèque simplement sous LGPL utilisée par un programme principal propriétaire est cependant aussi interdit).

> *5.e "The Application must not include software, documentation, or other materials that, in whole or in part, are governed by or subject to an Excluded License, or that would otherwise cause the Application to be subject to the terms of an Excluded License."*
>
> *1.l "“Excluded License” means any license requiring, as a condition of use, modification and/or distribution of the software subject to the license, that the software or other software combined and/or distributed with it be (i) disclosed or distributed in source code form; (ii) licensed for the purpose of making derivative works; or (iii) redistributable at no charge. Excluded Licenses include, but are not limited to the GPLv3 Licenses. For the purpose of this definition, “GPLv3 Licenses” means the GNU General Public License version 3, the GNU Affero General Public License version 3, the GNU Lesser General Public License version 3, and any equivalents to the foregoing."*

La réaction de la FSF se trouve à cette adresse : **http://www.fsf.org/blogs/licensing/windows-phone-gpl-ban**.

Nokia ou la stratégie erratique d'un géant

Tout à fait différemment de Google, Nokia a adopté bien avant, dès 2005, un système plus flexible avec Maemo (reconduit ensuite pour Meego), et malgré une diffusion volontairement difficile du N900 (un seul distributeur en France, jusqu'à trois mois d'attente pour un terminal !), le millionième téléchargement d'application a été fêté en mai 2010, sur le site *Garage* (**https://garage.maemo.org/**), site web d'hébergement qui regroupe les logiciels communautaires pour la plateforme (et dont il est vrai que les problèmes afférents à la diffusion massive ne sont jamais arrivés). En 2009, le finlandais Nokia a racheté le norvégien Trolltech, éditeur de la bibliothèque

graphique et de programmation en C++, Qt. Fonctionnant sur un système de double GPL/propriétaire[3] (obligeant dès lors tout développeur souhaitant diffuser du code fermé à s'acquitter de la licence commerciale), le code a immédiatement été relicencié en signant par là-même un abandon du *business model* historique, pour se recentrer en fait sur le service aux industriels. Ce pôle de services a été cédé en mars 2011 à une autre société finlandaise, Digia. Nokia a cédé à Digia l'exploitation des licences commerciales présentes (3500) et futures, ainsi que le support, tout en gardant le *copyright* et en continuant le développement.

Info

Sur Nokia et Qt, on pourra consulter les sites suivants :

- **http://qt.nokia.com/partners** ;
- **http://www.digia.com/C2256FEF0043E9C1/0/405002251** ;
- **http://blog.qt.nokia.com/2011/03/14/qt-and-digia-facts-and-fiction/**.

On peut supposer que Nokia a préféré pour le long terme refondre Maemo, fondé sur une bibliothèque concurrente issue du projet Gnome (Hildon est issu de GTK), en possédant le code même de Qt, c'est-à-dire les droits d'auteurs et le savoir faire. En effet, Qt a été la nouvelle base de MeeGo. Lancé en 2010 en partenariat avec Intel – qui développait Moblin de son côté –, MeeGo se voulant une fusion des deux environnements. Le projet a rapidement proposé par ailleurs un SDK pour que la communauté puisse facilement développer des applications, essentiellement sur tablette (**http://appdeveloper.intel.com/en-us/meego-sdk-suite?cid=Slim**).

Note

GTK et Qt sont historiquement opposés : ce sont les deux bibliothèques graphiques sur lesquelles reposent respectivement les environnements de bureau, sur le PC, que sont Gnome et KDE. Leurs philosophies sont très différentes et une certaine rivalité anime les communautés de libristes des deux bords.

Cependant, l'engagement de Nokia dans MeeGo est devenu des plus incertains depuis début 2011 (voir encadré ci-après), prémisse aux aléas qu'a connu la solution tout au long de l'année.

3. Ce type de licence entre en contradiction avec le fonctionnement communautaire : le droit d'auteur sur le code ne peut être détenu que par une seule personne physique ou morale ; toute personne voulant participer (par exemple pour corriger un *bug*) doit donc abandonner toute prétention sur le code. En réalité, on retrouve ce type de cession sur les logiciels du projet GNU comme GCC, à la différence du fait que le but recherché n'est pas d'ordre commercial.